

rag: un programma interattivo per analizzare rilevazioni relative a sciame di raggi cosmici

<http://didawiki.di.unipi.it/doku.php/fisica/informatica/>

Progetto di recupero del corso di Informatica Anno I e II (CDL
Fisica) 2015/16

Contents

1	Introduzione	1
1.1	Materiale in linea	2
1.2	Struttura del progetto e consegna	2
1.3	Valutazione del progetto	2
2	Il contesto: gli sciame di raggi cosmici	2
3	Il progetto	3
3.1	Sviluppo delle funzioni in <code>raggi.h</code>	4
3.2	Il programma <code>rag</code>	6
3.3	Comandi di <code>rag</code>	6
3.3.1	Aggiunta di un insieme di rilevazioni (<code>add</code>)	6
3.3.2	Cancellazione di un insieme di rilevazioni (<code>del</code>)	7
3.3.3	Calcolo approssimazione θ (<code>theta</code>)	7
3.3.4	Stampa rilevazione (<code>print</code>)	7
3.3.5	Eliminare il rumore (<code>setnoise</code>)	8
3.3.6	Verifica posizioni mancanti (<code>verify</code>)	8
3.3.7	Fine sessione (<code>exit</code>)	8
3.3.8	Aiuto in linea (<code>help</code>)	9
4	Codice e documentazione	9
4.1	Vincoli sul codice	9
4.2	Formato del codice	9
4.3	Relazione	10

1 Introduzione

Il progetto consiste nello sviluppo di un programma C che analizza le rilevazioni relative a sciame di raggi cosmici attraverso una semplice interfaccia.

1.1 Materiale in linea

Tutto il materiale si trova alla pagina web del corso, seguendo il link appropriato (primo anno o secondo anno).

Eventuali chiarimenti possono essere richiesti consultando i docenti l'orario di ricevimento e per posta elettronica.

1.2 Struttura del progetto e consegna

La consegna del progetto avviene *esclusivamente* per posta elettronica, attraverso il target **consegna** del **Makefile** contenuto nel kit di sviluppo del progetto (scaricabile dalla pagina degli assegnamenti) e deve contenere: tutti i file sorgente (**raggi.c** e **rmain.c**), la relazione in formato PDF e un file **gruppo.txt** che specifica numero di matricola, cognome, nome ed indirizzo di mail degli autori (max 2), secondo il formato (notare la virgola come separatore)

Zini, Gino, 123456, zini@madoc.it
Bixio, Nino, 456378, bixio@gmail.com

secondo le indicazioni del README nel kit di progetto. Tutti i file del progetto si devono trovare nella directory **RAGGICOSMICI/** e non sono ammesse sottodirectory.

*I progetti che non rispettano il formato o non consegnati con il target **consegna** non verranno accettati.*

1.3 Valutazione del progetto

Al progetto viene assegnato un punteggio da 0 a 30 in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice (suddivisione in moduli, uso di makefile e librerie etc.)
- efficienza e robustezza del software
- aderenza alle specifiche
- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione

La discussione del progetto in sede di orale tenderà a stabilire se lo studente è realmente l'autore e verterà su tutto il programma del corso. Il voto dello scritto + orale (ancora da 0 a 30) fa media con la valutazione del progetto per delineare il voto finale.

2 Il contesto: gli sciami di raggi cosmici

La Terra¹ è continuamente bombardata da raggi cosmici²: particelle cariche provenienti dal cosmo. Alcune di esse, di energia molto elevata, quando interagiscono negli strati alti dell'atmosfera producono sciami di particelle secondarie

¹Per qualsiasi dubbio di natura fisica fate riferimento a Carmelo Sgrò e Luca Baldini

²https://it.wikipedia.org/wiki/Raggi_cosmici

che arrivano a terra come un fronte compatto largo anche decine di chilometri. Possiamo schematizzare il fronte come un piano ortogonale alla direzione di arrivo della particella primaria.³ Per studiare questi raggi cosmici si dispongono tanti piccoli rivelatori a terra, in modo da formare un array abbastanza regolare. In questo esercizio studieremo un caso semplificato (ma non troppo irrealistico) della tecnica per ricostruire la direzione di arrivo dello sciame.

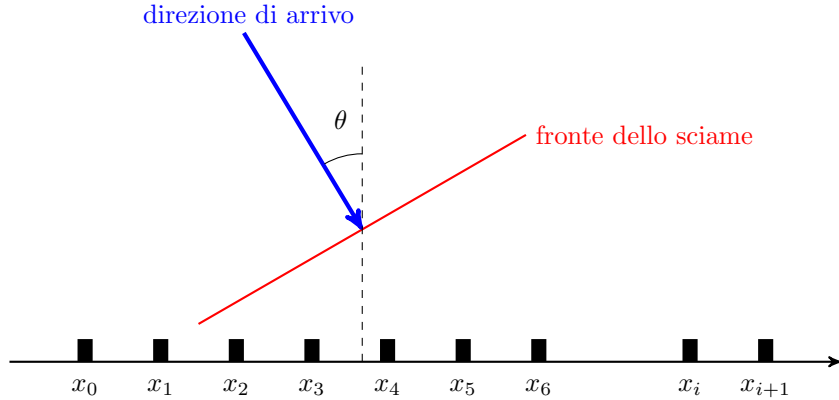


Figure 1: Schema semplificato, in una dimensione, di rivelatore di sciame estesi di raggi cosmici. Un array di rivelatori è posto a terra (rettangoli neri) alle coordinate x_i . Una particella cosmica primaria che arriva con angolo zenith θ produce uno sciame il cui fronte è rappresentato da un piano (linea rosso). Quando il fronte raggiunge un rivelatore questo ne registra il tempo di passaggio. Combinando i tempi di due (o più) rivelatori si ricostruisce l'angolo θ .

Lo schema appena descritto è mostrato in figura 1. Quando una particella di alta energia arriva con un angolo θ , fa scattare i rivelatori a terra (rettangoli neri in figura 1) che registrano il tempo assoluto di arrivo del fronte dello sciame. Con un po' di semplice geometria si ricava la relazione tra i tempi di arrivo in due rivelatori i e j , la loro posizione e l'angolo θ :

$$t_i - t_j = \frac{x_i - x_j}{v} \sin(\theta) \quad (1)$$

con v velocità delle particelle (in pratica la velocità della luce $v = 299792458$ m/s)

3 Il progetto

Lo scopo del progetto è lo sviluppo un programma C per l'analisi di rilevazioni relative ai raggi cosmici e della relativa interfaccia d'uso.

Il programma mette a disposizione un insieme di operazioni (caricamento di un nuovo file di rilevazioni, analisi del rumore, analisi delle rilevazioni mancanti, calcolo di θ etc.) assieme ad alcune funzioni di utilità come il salvataggio su file di un insieme di rilevazioni elaborato, la loro visualizzazione su stdout etc).

³In realtà il fronte è leggermente curvato, ma ciò è irrilevante per lo scopo di questo esercizio.

L'interfaccia fra l'utente ed il programma e' un semplice *interprete di comandi*: una volta attivato il programma interagisce con l'utente tramite una linea di comando digitata da tastiera. In particolare, si mette in attesa di comandi da parte dell'utente, esegue le richieste e visualizza i risultati.

Si assume inoltre che in ogni momento l'utente possa lavorare al massimo su 3 insiemi di rilevazioni diverse ciascuno rappresentato da una lista concatenata di valori secondo il tipo definito in **raggi.h**.

Lo studente deve prima realizzare nel file **raggi.c** le funzioni il cui prototipo si trova nel file **raggi.h** seguendo la traccia nella Sezione 3.1 e poi realizzare l'interfaccia ed la funzione **main** (nel file **rmain.c**) seguendo le specifiche della Sezione 3.2, utilizzando per portare a termine le varie operazioni le funzioni gia' sviluppate.

Ricordiamo anche di leggere il file README nella directory RAGGICOS-MICI che contiene le indicazioni precise su come sviluppare ed eseguire il progetto.

3.1 Sviluppo delle funzioni in **raggi.h**

Esercizio 1 Il file *dataPR1.txt* contiene alcune rilevazioni (chiamate *hit* in gergo) relative ad un singolo "evento", ovvero le posizioni e i tempi di arrivo dei vari rivelatori. Leggere attentamente il file **test_one.c** ed il file **raggi.h** e realizzare le funzioni utilizzate dal test (**leggi_registrazione**, **inserisci**, etc...) utilizzando le strutture dati definite in **raggi.h**. Eseguire correttamente

```
make test1
```

Esercizio 2. Utilizzando sempre il file *dataPR1.txt*, implementare le funzioni che realizzano l'ordinamento per tempo e posizione (leggere attentamente il file **test_two.c**) ed eseguire il test per effettuare l'ordinamento dei dati

```
make test2
```

Esercizio 3. Eseguire, sempre utilizzando il file *dataPR1*,

```
make test3
```

per testare l'inserimento ordinato di un elemento alla volta (leggere attentamente il file **test_three.c**) .

Esercizio 4. Eseguire

```
make test4
```

per testare l'inserimento ordinato sul file **dataPR.txt** che contiene tutte le rilevazioni relative all'evento in esame (leggere attentamente il file **test_four.c**)

.

Esercizio 5. Un modo semplice⁴ per ricostruire l'angolo θ è da due rivelatori, invertendo la formula 1. Il nostro obiettivo è stimare θ e, partendo dalla lista ordinata per tempo, attraverso una funzione che calcola θ eseguita su tutte le coppie di elementi successivi. I tempi di arrivo sono soggetti a fluttuazioni casuali legati alla risoluzione dei rivelatori; si calcoli media e deviazione standard di θ , per stimare la direzione di arrivo della particella primaria e la sua incertezza. Sempre con riferimento al file `dataPR.txt`, analizzare il file `test_five.c` implementando le funzioni utilizzate (`compute_theta`, `media`, etc...) ed eseguire

```
make test5
```

per ottenere media e varianza dell'angolo theta ottenuto elaborando le rilevazioni. [Nota: In fig 2 si notano le 2 popolazioni, delle θ fatte con hit buone e non a partire dal file `dataPR.txt`. Nel nostro esperimento i dati con rumore sono già etichettati ed esclusi dal test.]

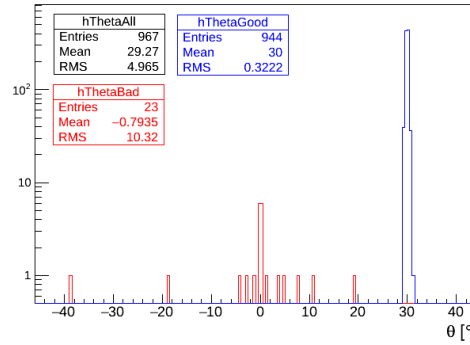


Figure 2: Distribuzione dei θ ricostruiti a partire dal file dei dati originale.

Esercizio 6. Sfruttando il fatto che i rivelatori sono equispaziati, e che esistono almeno due rilevazioni contigue nel file dei dati, determinare la distanza fra i rivelatori e stabilire se mancano delle rilevazioni e da quali rivelatori. Sempre con riferimento al file `dataPR.txt`, analizzare il file `test_six.c` implementando la funzione `rilevazioni_mancanti` che restituisce la lista degli intervalli con rilevazioni mancanti. Il tipo della lista delle rilevazioni mancanti è definito in `raggi.h` insieme alle funzioni che la manipolano. Realizzare le funzioni utilizzate in `test_six.c` e verificare che il sesto test dia esito positivo con

```
make test6
```

Esercizio 7. Si vuole adesso identificare le hit di rumore e quindi non dovuti alla particella primaria. Ci sono diversi modi per farlo. Nel nostro caso è noto che la risoluzione del sistema (stimata da altri) è sempre migliore di 2° (2 gradi). Quindi se una hit è dovuta al rumore il θ ottenuto associandola ad una hit buona sarà sicuramente “sbagliato”, ovvero disterà più di 2° dal valore centrale di θ .

⁴Il modo ottimale, che combina tutti i rivelatori insieme, non è rilevante allo scopo di questo esercizio.

Quindi è possibile procedere come segue: trovare una coppia di hit sicuramente buona e utilizzare una delle due rilevazioni della hit individuata combinandola con le rilevazioni rimanenti. Usando questa strategia (o una equivalente) implementare la funzione `verifica_rilevazioni` che analizza tutte le rilevazioni di una certa lista e ne modifica il campo `flag` assegnando i valori OK o NOISE dopo aver verificato se si tratta di una rilevazione di rumore o meno. Analizzare `test_seven.c`, realizzare le funzioni utilizzate dal test e verificare che l'esito del settimo test sia corretto con

```
make test7
```

3.2 Il programma rag

Il programma `rag` viene attivato digitando

```
bash:~$ ./rag
```

Una volta attivato, visualizza un opportuno prompt

```
bash:~$ ./rag
Welcome in rag!
-?
```

ed entra in un ambiente interattivo che permette l'esecuzione di un insieme di comandi. Nella Sezione 3.3 per ogni comando accettato, è specificata la sintassi, il significato, e alcune delle condizioni in cui è richiesto di segnalare un errore. Ulteriori condizioni di errore devono essere fissate dallo studente durante la stesura del progetto e specificate nella documentazione.

3.3 Comandi di rag

L'interprete di `rag` accetta dall'utente l'insieme di comandi descritti nel seguito. I comandi vanno utilizzati uno alla volta e non sono annidabili.

3.3.1 Aggiunta di un insieme di rilevazioni (add)

Comando add:

```
add pathfile
```

Descrizione:

Carica da file un nuovo insieme di rilevazioni allocando la lista corrispondente.

Errori:

Se sono già presenti 3 insiemi di rilevazioni, se il file non esiste o non può essere aperto, se avviene un errore durante la creazione della lista.

Esempio:

```
-? add pippo.txt
Insieme caricato correttamente in posizione 2
-? add ciccio.txt
Errore: sono già presenti tre insiemi di rilevazioni
```

3.3.2 Cancellazione di un insieme di rilevazioni (del)

Comando del:

`del i`

Descrizione:

La lista *i*-esima (num può essere uguale a 0, 1 e 2) viene cancellata dall'insieme di lavoro.

Errori:

Se l'insieme di lavoro non è presente.

Esempio:

```
-? del 0
Cancellazione eseguita con successo
-? del 0
Errore: rilevazione 0 non presente
```

cancella la rilevazione 0 e segnala un errore alla seconda richiesta.

3.3.3 Calcolo approssimazione θ (theta)

Comando theta:

`theta i`

Descrizione:

Se la lista *i* è presente nell'insieme calcola l'approssimazione di θ ordinando la lista per tempo, applicando l'equazione 1 e calcolando la media e la deviazione standard dei valori ottenuti. Il risultato viene mostrato sullo standard output.

Errori:

La lista non è presente etc

Esempio:

```
-? theta 0
Media = 29.268606909751437, Varianza = 4.967922600360287
-? theta 2
Errore: rilevazione 2 non presente
```

calcola l'approssimazione correttamente e segnala un errore di rilevazione non presente.

3.3.4 Stampa rilevazione (print)

Comando print:

`print i [pathfile]`

Descrizione:

Il secondo parametro (*pathfile*) è opzionale. Con un solo parametro visualizza il contenuto della rilevazione *i* (utilizzando la funzione fornita dai docenti) su standard output, con due parametri lo scrive sul file *pathfile* (sovrascrivendolo completamente se già presente).

Errori:

La lista non è presente, il file non può essere creato/sovrascritto etc

Esempio:

```
-? print 0 ciccio.txt  
Scrittura effettuata correttamente
```

3.3.5 Eliminare il rumore (setnoise)**Comando setnoise:**

```
cleannoise i
```

Descrizione:

setta il flag degli elementi verificando se sono attendibili (OK) o se sono rumore (NOISE) :

Esempio:

```
-? setnoise 1  
Errore: rilevazione 1 non presente  
-? setnoise 0  
Rilevazione 0 etichettata
```

3.3.6 Verifica posizioni mancanti (verify)**Comando verify:**

```
verify i
```

Descrizione:

Verifica se mancano delle rilevazioni nell'insieme *i* (considerando i rilevatori equispaziati) e stampa la lista delle rilevazioni mancanti (con la funzione fornita dai docenti). La lista deve essere ordinata per posizione.

Esempio:

```
-? verify 0  
<< begin 2 rilevazioni>> << 70000.0, 2>> << 95000.0, 3>> << end  
>>
```

3.3.7 Fine sessione (exit)**Comando exit:**

```
exit
```

Descrizione:

Provoca la terminazione del programma. Si deve chiedere conferma all'utente.

Esempio:

```
-? exit  
Chiusura (y/n) ?
```


3.3.8 Aiuto in linea (help)

Comando:

help

Descrizione:

Visualizza l'elenco dei comandi disponibili e la loro sintassi, descrivendo brevemente il loro significato.

4 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

4.1 Vincoli sul codice

Il codice consegnato deve rispondere ai seguenti vincoli:

- il codice deve utilizzare la struttura dati definita nel file `raggi.h` contenuto nel kit;
- il codice delle funzioni definite in `raggi.h` deve superare tutti i test previsti nel Makefile contenuto nel kit;
- il codice deve compilare senza errori o warning gravi utilizzando le opzioni `-Wall -pedantic`
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)

4.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- un'intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore; firma dell'autore.
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali non di uso ovvio, che spieghi a cosa servono

4.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 10 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione *NON deve ripetere le presenti specifiche ma deve rendere comprensibile il lavoro svolto per realizzarle ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli*. In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali) e loro motivazioni
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- le difficoltà incontrate e le soluzioni adottate
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- istruzioni per l'utente su come compilare/eseguire ed utilizzare il codice (in quale directory deve essere attivato, quali sono le assunzioni fatte etc) (da riportare anche nel README)

La relazione deve essere in formato ".pdf".